Ollscoil na Gaillimhe
University of Galway

CT336/CT404
Graphics & Image Processing

4th year B.Sc. (CS&I.T.).
3rd/4th year B.Sc. (Maths-Computing)
4th year B.Sc. Biomedical Science?
Erasmus/visiting students?

University
*of*Galway.ie

Module Introduction

Ollscoil na Gaillimhe
University of Galway

- Module Syllabus Updates

- Image Graphics is covered in other modules at present:
  - CT3536 Games Programming (in 3rd year) - using Unity.
  - CT255 Next Generation Technologies (2nd year) - 2D games in Java

- Handout

- Some prior knowledge of programming is required. We'll be using mostly JavaScript in Graphics part.

- Refer to the book (CG) Appendix A for a quick overview of JavaScript

Module Introduction

Ollscoil na Gaillimhe
University of Galway

COMPUTER GRAPHICS:

- processing and display of images of objects that exist conceptually rather than physically
- emphasis on the generation of an image from a model of the objects, illumination, etc.
- emphasis on real-time rendering of images
- Ideas from 2D graphics extend to 3D graphics

DIGITAL IMAGE PROCESSING/ANALYSIS:

- processing and display of images of real objects
- emphasis on the modification and/or analysis of the image in order to automatically/semi-automatically extract useful information
- Processing leads to more advanced feature extraction and pattern recognition techniques for image analysis and understanding

A Reflection regarding Exams

Ollscoil na Gaillimhe
University of Galway

“A lot of people give far too little detail in these questions, and/or don't address the discussion parts - they just give some high-level definitions and consider it done -- which isn't enough for final year undergrad, and isn’t answering the question.

More is expected in answers than just repeating what's in my slides.

The top performers demonstrate a higher level of understanding and synthesis as well as more detail about techniques and discussion of what they do on a technical level and how they fit together”

Lecture 1: Introduction to 2D Graphics

Ollscoil na Gaillimhe
University of Galway

- What are Images?
- Graphics Pipeline
- Graphics Libraries
- 2D vector graphics
- 2D Transformations Graphics hardware
- 2D raster graphics
- Introduction to HTML5/Canvas for applied 2D graphics

Digital Images - Bitmaps

- Bitmaps: grid-based arrays of colour or brightness (greyscale) information
- Pixels (picture elements): the cells of a bitmap
- Depth: bits-per-pixel (bpp)

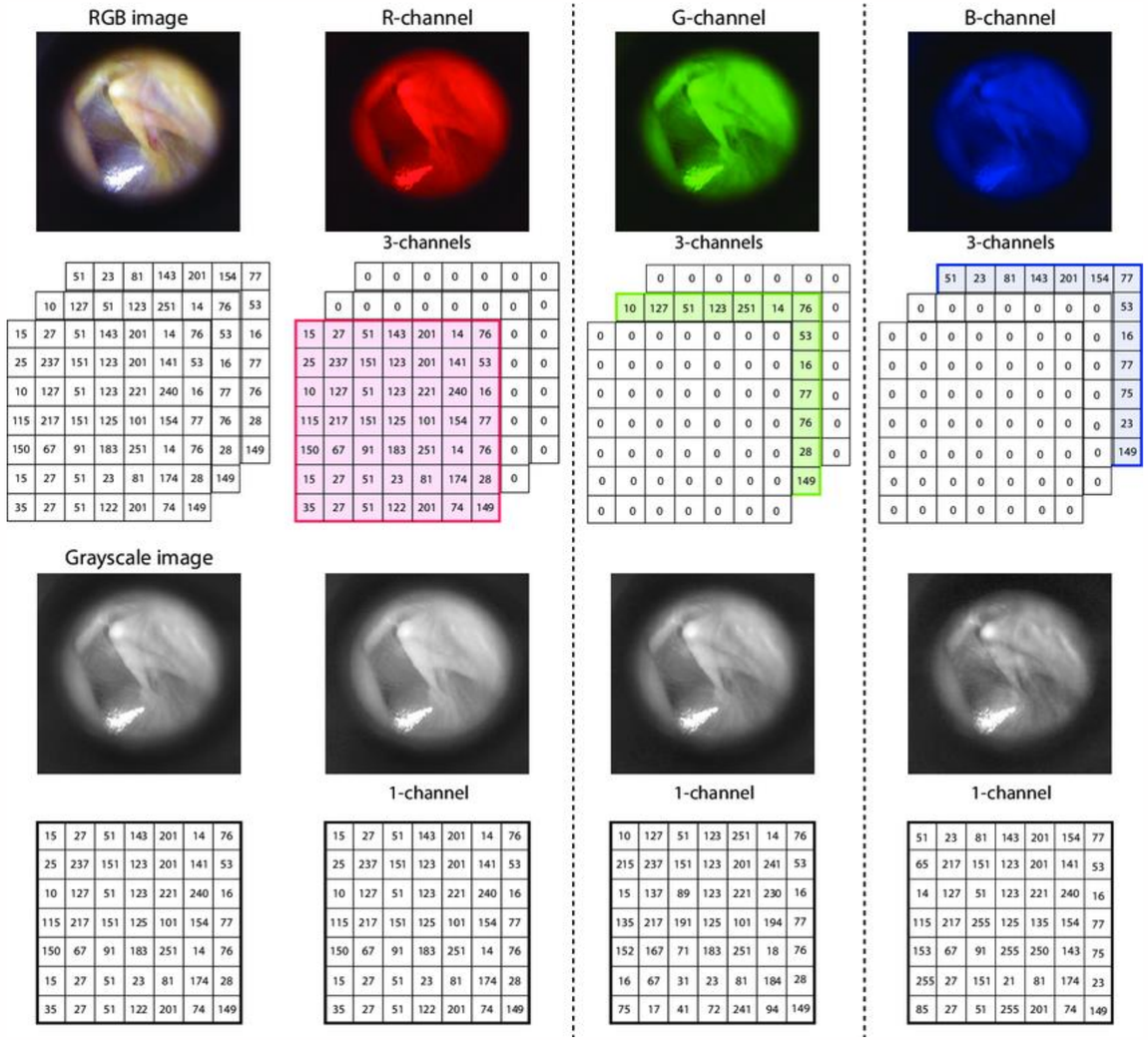


Image reproduced from [1]

[1] Color Dependence Analysis in a CNN-Based Computer-Aided Diagnosis System for Middle and External Ear Diseases - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Matrix-representation-of-a-digital-image-upper-row-from-left-to-right-image-in-the_fig2_359806413 [accessed 11 Sept 2024]

Colour Encoding Schemes

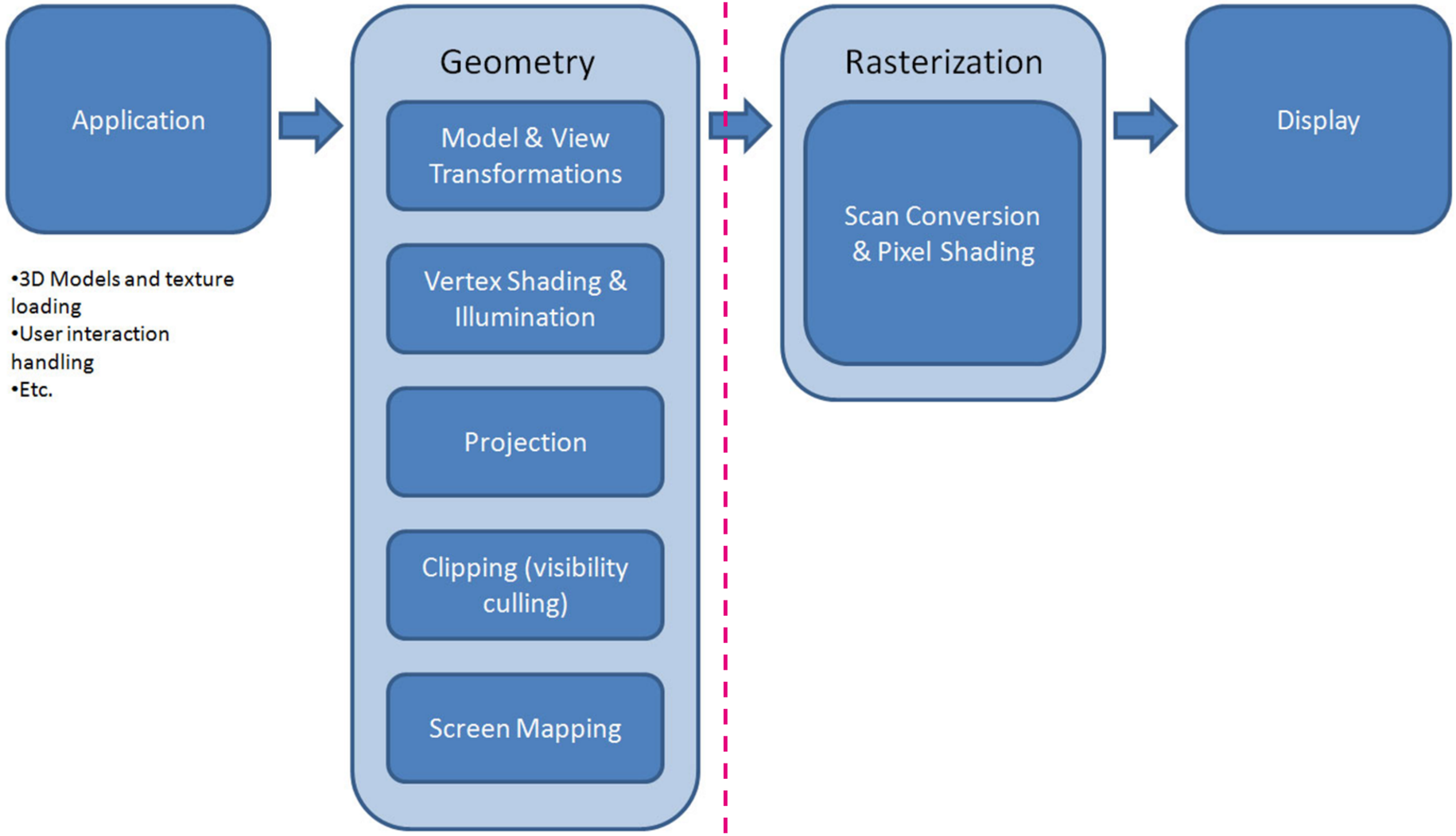
- Colour is most commonly represented using the RGB scheme
- Other schemes, such as HSI, although superior to RGB, are not as commonly used
- Greyscale images encode brightness values, or scales of grey, rather than colours

R: 114, G: 64, B: 189

R: 255, G: 0, B: 0
H: 0, S: 255, I: 255

R: 255, G: 158, B: 158
H: 0, S: 97, I: 255

Real-Time Graphics Pipeline



- 3D Models and texture loading
- User interaction handling
- Etc.

(World/obj space algs.)

(Image space algs.)

Graphics Software

Ollscoil na Gaillimhe
University of Galway

https://www.khronos.org/opengl/

- GPU (Graphical Processing Unit)
  - Internal, fast-accessed GPU memory
  - Parallel processors for vertices and fragments for faster graphics renderings
  - Included in modern computers to complement CPU

- OpenGL
  - 2D and 3D graphics API existing since 1992
  - Supported by graphics hardware (GPU's) in most computing devices today
  - WebGL: 3D graphics on the web **(check https://get.webgl.org/ for your browser's compatibility)**
  - OpenGL ES for 'Embedded Systems' such as tablets and mobile phones
  - Being replaced by newer API's e.g. Vulkan, Metal, Direct3D. WebGL replaced by WebGPU
  - Previously a client/server system: CPU+Application as a client sending commands/data and GPU as a server
  - Later replaced by 'programmable' graphics interface (OpenGL 3.0) to write GPU programs (shaders) to be run by GPU directly

Graphics Formats

Ollscoil na Gaillimhe
University of Galway

- **Vector Graphics**: images described in terms of co-ordinate drawing operations (AutoCAD, PowerPoint, Flash, SVG)
- File format SVG (Scalable Vector Graphics): image is specified by vectors which are scalable without losing any quality

- **Raster Graphics**: images described as pixel-based bitmaps (Photoshop, Paint Shop Pro, HTML5/Canvas)
- File formats are GIP, PNG, JPEG: image is specified by storing color values for each pixel

2D Vector Graphics

Ollscoil na Gaillimhe
University of Galway

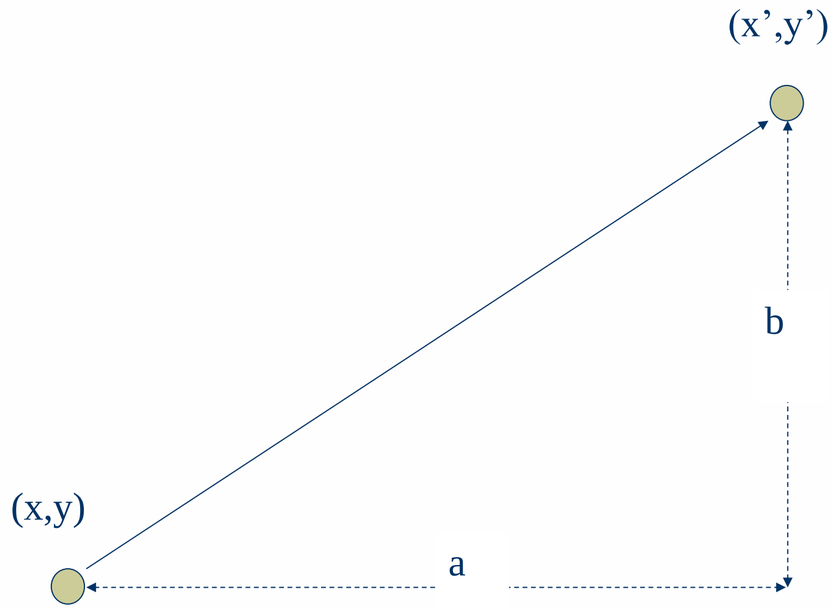
- 2D vector graphics describe drawings as a series of instructions related to a 2-dimensional co-ordinate system.
- Any point in this co-ordinate system can be specified using two numbers (x,y):
  - a horizontal component – usually called ‘x’ and measuring the distance from the left hand edge of the screen or window
  - a vertical component – usually called ‘y’ and measuring the distance from the bottom of the screen/window (although, sometimes measuring the distance from the top!)

- 2D Vector Transformations
  - Translation
  - Rotation
  - Scaling
  - (Using Matrix notation)
  - Order of transformations

2D Translation

- Translation in 2 dimensions: movement of a point *(x,y)* to some other point *(x', y')*

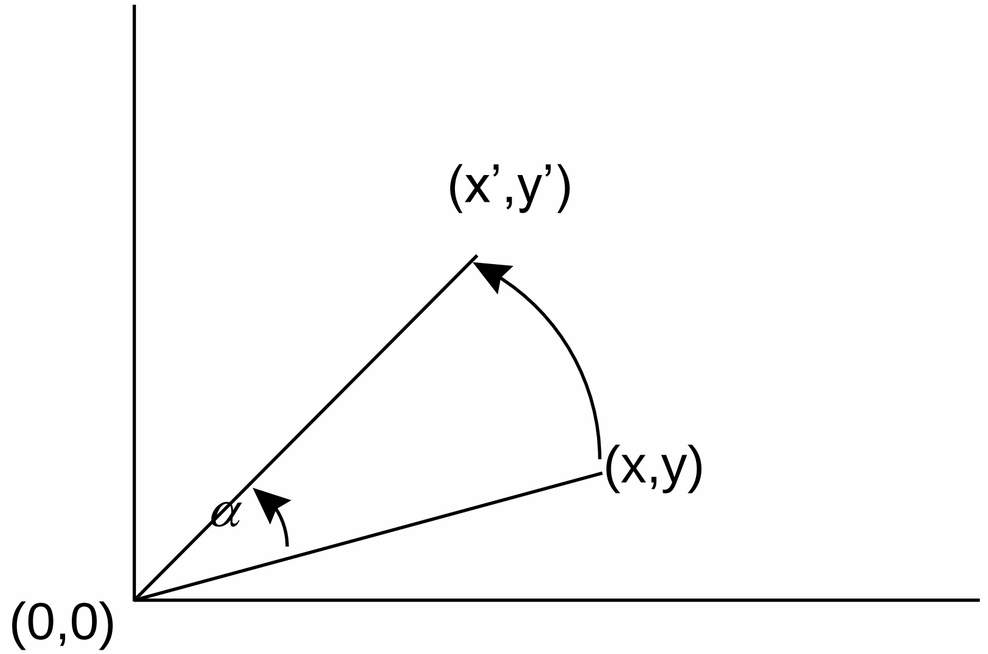
$$x' = x + a$$
$$y' = y + b$$



2D Rotation *of a point*

- Rotation about the origin (0,0) is simplest:

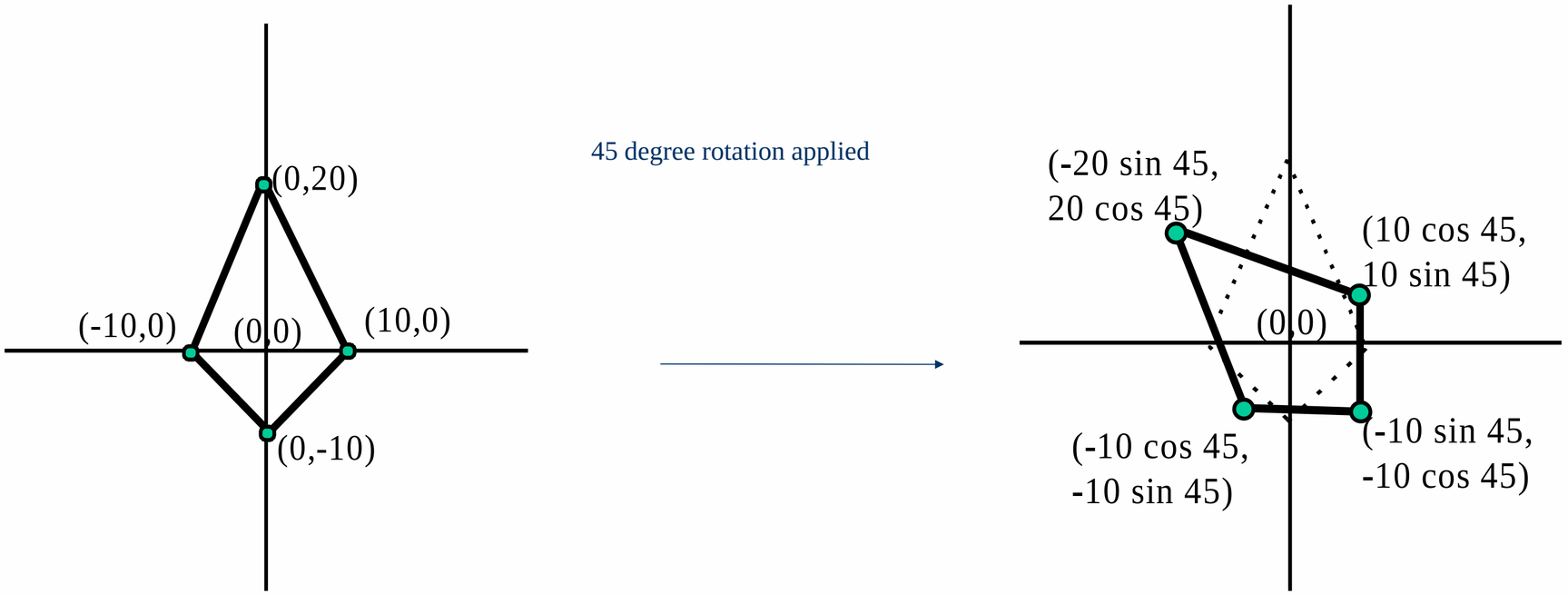
$$x' = x \cos \alpha - y \sin \alpha$$
$$y' = x \sin \alpha + y \cos \alpha$$



2D Rotation *of an Object*

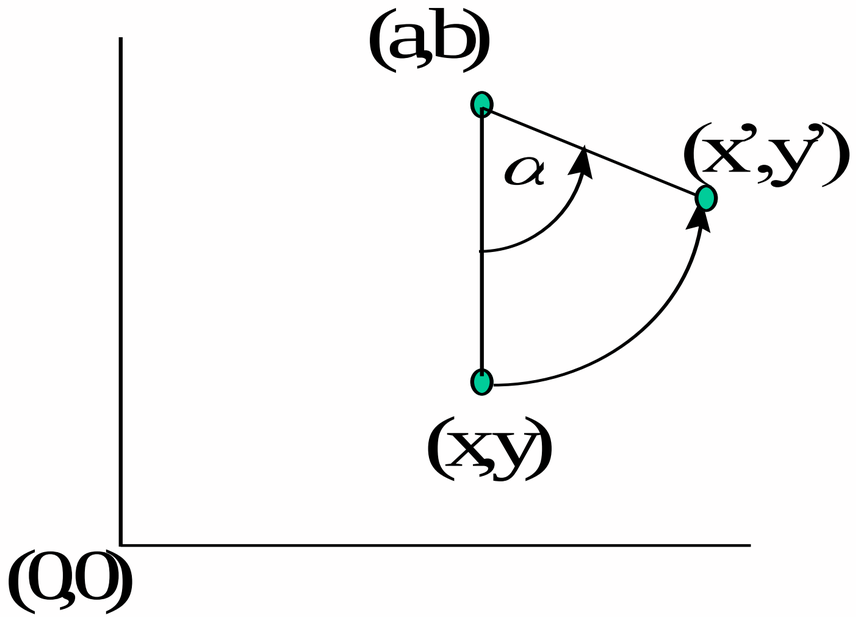
Ollscoil na Gaillimhe
University of Galway

- In vector graphics, objects are defined as a series of drawing operations (e.g. straight lines) performed on a set of vertices
- To rotate a line or more complex object, simply apply the equations to the (x,y) co-ordinates of each vertex



Arbitrary 2D Rotation

In order to rotate around an arbitrary point (a,b), we perform translation, then rotation, then reverse the translation



$$x' = a + (x - a) \cos \alpha - (y - b) \sin \alpha$$
$$y' = b + (x - a) \sin \alpha + (y - b) \cos \alpha$$

Matrix Notation

- Matrix notation is commonly used for vector graphics:
  - more complex operations are easier in matrix format
  - several operations can be combined easily into one matrix using matrix algebra

$$[x' \;\; y'] = [x \;\; y] \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

rotation about (0,0)

$$[x' \;\; y' \;\; 1] = [x \;\; y \;\; 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{bmatrix}$$

translation

Scaling

- Scaling of an object is achieved by considering each of its vertices in turn, and multiplying its x and y values by the scaling factor.

- A scaling factor of 2 will double the size of the object, while a scaling factor of 0.5 will halve it

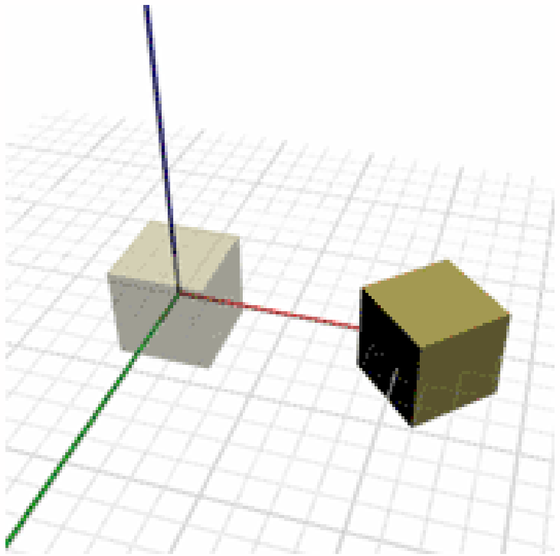
- It is possible to have different scaling factors for x and y, resulting in a *stretch*:

  $x' = x * s$

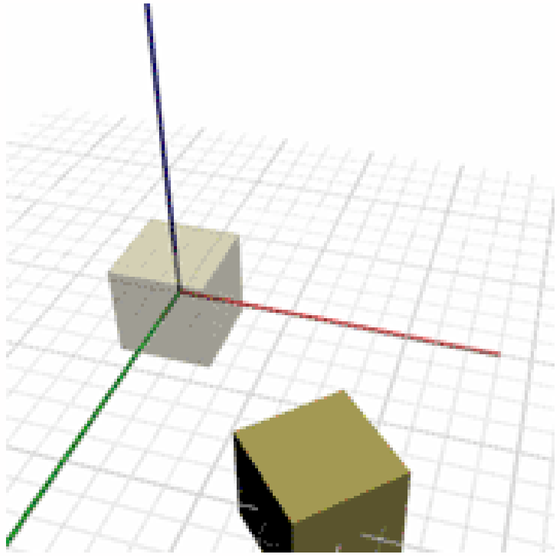
  $y' = y * t$

- If the object is not centred on the origin, then scaling it will also effect a translation

Order of Transformations



- Translation 2 units along the red axis
- Then Rotation by 45 degrees around the (new) centre



- Rotation by 45 degrees
- Then Translation 2 units along the (rotated) red axis

2D Raster Graphics

Ollscoil na Gaillimhe
University of Galway

The raster approach to 2D graphics considers digital images to be grid-based arrays of pixels, and operates on the images at the **pixel level.**

We will discuss the nature of raster graphics via some of its specific issues and techniques during our exercises with Canvas2D.

Introduction to HTML5/Canvas

Ollscoil na Gaillimhe
University of Galway

- HTML – HyperText Markup Language: page-description language used for websites
- HTML5: brings major updates and improvements to the power of client-side web development

- Canvas : a 2D raster graphics component in HTML5 (https://www.w3schools.com/html/html5_canvas.asp)
- Canvas with 3D (WebGL): a 3D graphics component in HTML5, more likely to be hardware accelerated (but also more complex)

- In this module we'll explore both the 2D and 3D rendering contexts of Canvas
- The 3D context is based on *OpenGL* and is called 'WebGL'

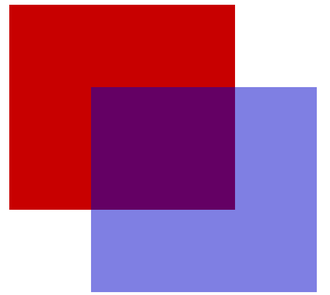
Some useful libraries exist for dealing with Canvas 2D/3D, e.g. Easeljs, Pixijs, Threejs..

Canvas: Rendering Contexts

Ollscoil na Gaillimhe
University of Galway

- <canvas> creates a fixed size drawing surface that exposes one or more “rendering contexts”. The getContext() method returns an object with tools (methods) for drawing.

```
<html>
 <head>
  <script>
    function draw() {
      var canvas = document.getElementById("canvas");
      var ctx = canvas.getContext("2d");
      ctx.fillStyle = "rgb(200,0,0)";
      ctx.fillRect (10, 10, 55, 50);
      ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
      ctx.fillRect (30, 30, 55, 50);
}
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

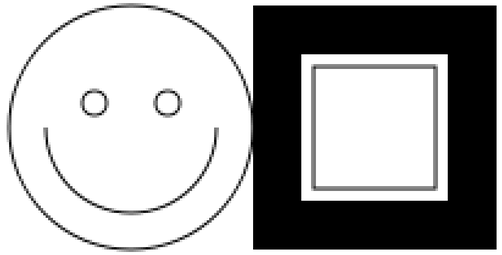


examples/canvasExample1.html

Canvas2D: primitives

- Canvas2D only supports one primitive shape - rectangles. All other shapes must be created by combining one or more *paths*.
- Luckily, we have a collection of path drawing functions which make it possible to compose complex shapes.

```
function draw(){
  var canvas = document.getElementById('canvas');
  var ctx = canvas.getContext('2d');

  ctx.fillRect(125,25,100,100);
  ctx.clearRect(145,45,60,60);
  ctx.strokeRect(150,50,50,50);

  ctx.beginPath();
  ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
  ctx.moveTo(110,75);
  ctx.arc(75,75,35,0,Math.PI,false);   // Mouth (clockwise)
  ctx.moveTo(65,65);
  ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
  ctx.moveTo(95,65);
  ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
  ctx.stroke(); // renders the Path that has been built up..
}
```

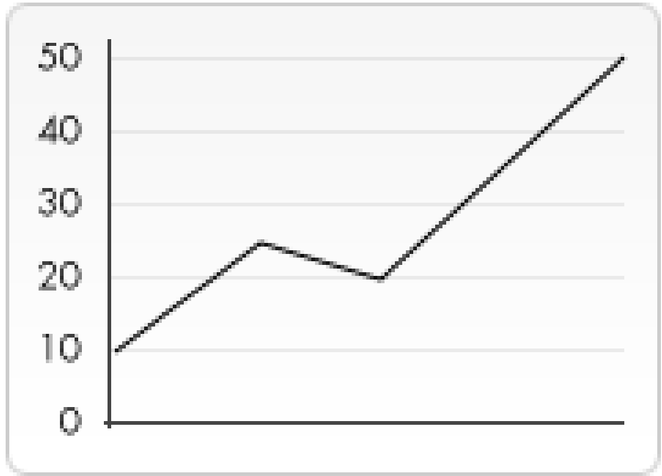


canvasExample2.html

Canvas2D: drawImage

This example uses an external image as the backdrop of a small line graph

```
function draw() {
  var ctx = document.getElementById('canvas').getContext('2d');
  var img = new Image();
  img.src = 'backdrop.png';
  img.onload = function(){
    ctx.drawImage(img,0,0);
    ctx.beginPath();
    ctx.moveTo(30,96);
    ctx.lineTo(70,66);
    ctx.lineTo(103,76);
    ctx.lineTo(170,15);
    ctx.stroke();
  }
}
```



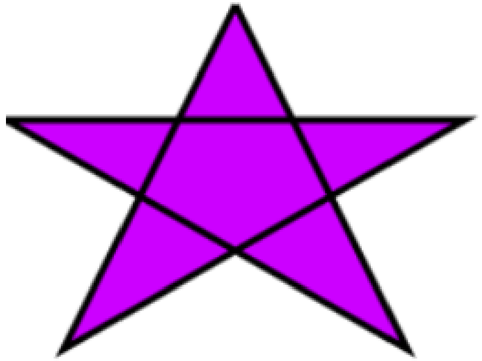
canvasExample3.html

Canvas2D: Fill and Stroke colours

Ollscoil na Gaillimhe
University of Galway

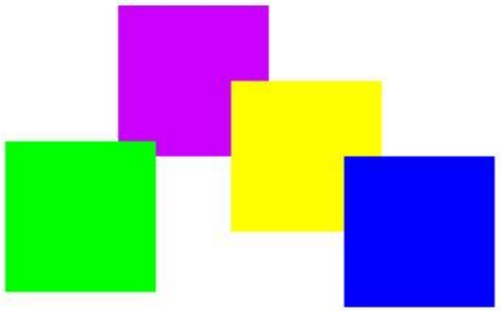
```
<html>
 <head>
  <script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

  // Filled Star
  context.lineWidth=3;
  context.fillStyle="#CC00FF";
  context.strokeStyle="#ffff00"; // NOT lineStyle!
  context.beginPath();
  context.moveTo(100,50);
  context.lineTo(175,200);
  context.lineTo(0,100);
  context.lineTo(200,100);
  context.lineTo(25,200);
  context.lineTo(100,50);
  context.fill(); // colour the interior
  context.stroke(); // draw the lines

}
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="300" height="300"></canvas>
 </body>
</html>
```

Colours can be specified by names (“red”), a string of the form “rgb(r,g,b)” or hexadecimal color codes “#RRGGBB”.



canvasExample5.html

Canvas2D: Translations



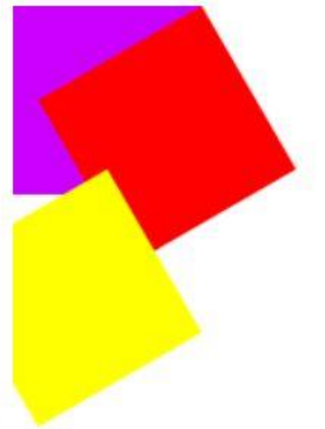
```
<html>
 <head>
  <script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');
  context.save(); // save the default (root) co-ord system

  context.fillStyle="#CC00FF"; // purple
  context.fillRect(100,0,100,100);

  // translates from the origin, producing a nested co-ordinate system
  context.translate(75,50);
  context.fillStyle="#FFFF00"; // yellow
  context.fillRect(100,0,100,100);

  // transforms further, to produce another nested co-ordinate system
  context.translate(75,50);
  context.fillStyle="#0000FF"; // blue
  context.fillRect(100,0,100,100);

  context.restore(); // recover the default (root) co-ordinate system
  context.translate(-75,90);
  context.fillStyle="#00FF00"; // green
  context.fillRect(100,0,100,100);
}
 </script>
 </head>
```

These coordinate systems are nested

```
<body onload="draw();">
  <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

canvasTranslateExample.html

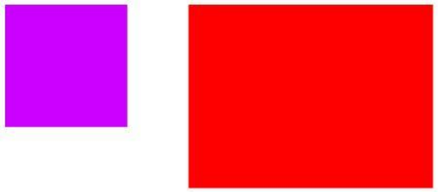
Order of Tansformations



```
<html>
 <head>
  <script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');
  context.save(); // save the default (root) co-ord system

  context.fillStyle="#CC00FF"; // purple
  context.fillRect(0,0,100,100); // positioned with TL corner at 0,0

  // translate then rotate
  context.translate(100,0);
  context.rotate(Math.PI/3);
  context.fillStyle="#FF0000"; // red
  context.fillRect(0,0,100,100); // positioned with TL corner at 0,0

  // recover the root co-ord system
  context.restore();
  // rotate then translate
  context.rotate(Math.PI/3);
  context.translate(100,0);
  context.fillStyle="#FFFF00"; // yellow
  context.fillRect(0,0,100,100);  // positioned with TL corner at 0,0
}
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

canvasOrderOfTransformsExample.html

Scaling



```
<html>
 <head>
  <script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

  context.fillStyle="#CC00FF"; // purple
  context.fillRect(0,0,100,100); // positioned with TL corner at 0,0

  context.translate(150,0);
  context.scale(2,1.5);
  context.fillStyle="#FF0000"; // red
  context.fillRect(0,0,100,100); // positioned with TL corner at 0,0

}
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

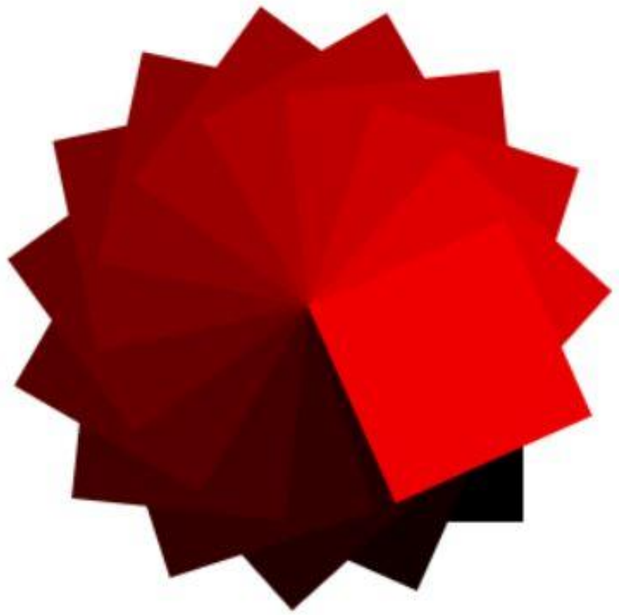
canvasScaleExample.html

Canvas2D: Programmatic graphics

Ollscoil na Gaillimhe
University of Galway

```
<html>
 <head>
  <script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');

  context.translate(150,150);

  for (i=0;i<15;i++) {
            context.fillStyle = "rgb("+(i*255/15)+",0,0)";
            context.fillRect(0,0,100,100);
            context.rotate(2*Math.PI/15);
  }

}
  </script>
 </head>

 <body onload="draw();">
   <canvas id="canvas" width="600" height="600"></canvas>
 </body>
</html>
```

context.fillStyle="rgb(200,0,0)"



canvasRotateLoopExample.html

Ollscoil na Gaillimhe
University of Galway

Thank *you*

University
*of*Galway.ie